Práctica 1: Pacman 1.0

Curso 2020-2021. Tecnología de la Programación de Videojuegos 1. UCM

Fecha de entrega: 20 de noviembre de 2020

El objetivo de esta práctica es implementar en C++/SDL una versión simplificada del conocido juego Pacman creado por Toru Iwatani en 1980 (véase `https://es.wikipedia.org/wiki/Pac-Man`). Se utilizará la librería SDL para manejar toda la entrada/salida del juego. Tomaremos como base para la mecánica del juego la versión del Pacman disponible para jugar online en `http://www.webpacman.com/`. A continuación se detallan las principales simplificaciones y diferencias que nuestro juego tendrá respecto al original indicado:

- El mapa del juego se visualizará de manera diferente y simplificada. En particular usaremos la misma textura para todos los muros. Además no habrá compuerta en la guarida de los fantasmas.
- El juego acaba cuando el pacman se come toda la comida (el jugador gana) o bien cuando el pacman se queda sin vidas (el jugador pierde). No hay por tanto concepto de puntuación con las simplificaciones que ello conlleva, ni de paso de nivel. Se deja libertad respecto a la manera en la que se informa al usuario tanto del número de vidas restantes, como de si gana o pierde cuando el juego acaba (en consola, en ventana emergente o en la propia ventana SDL).
- No hay animaciones en el pacman ni en los fantasmas, es decir, las texturas del pacman y de los fantasmas son siempre las mismas. Se deja también libertad respecto a la forma en la que se le indica al usuario que el pacman tiene energía (es decir, que se come a los fantasmas que se encuentre). Además las velocidades de pacman y fantasmas son siempre las mismas.
- El movimiento y renderizado del pacman y de los fantasmas se realiza cada cierto tiempo y de manera discreta, es decir, cuando corresponde se pasa directamente de una casilla del mapa a la siguiente (sin pasar de forma gradual de la una a la otra).
- Cuando el pacman se come a un fantasma, éste vuelve directamente a su posición inicial (y no se va moviendo hacia ella como ocurre en el original).
- Los fantasmas no tienen inteligencia alguna y se mueven de manera aleatoria independientemente de donde se encuentren el pacman y los otros fantasmas.

Detalles de implementación

Formato de ficheros de mapas y configuraciones iniciales

El mapa y la configuración inicial de los personajes se lee de un fichero de texto con el siguiente formato: En la primera línea hay dos números enteros separados por un espacio que indican el número de filas y columnas respectivamente del mapa. A continuación viene el contenido del mapa incluyendo las posiciones iniciales de los fantasmas y el pacman. Cada fila del mapa se representa en una línea que contiene números enteros separados mediante espacios. Cada número representa el contenido de la celda correspondiente a la posición en la que se encuentra. Los valores enteros posibles y sus correspondientes significados son los siguientes: 0 = vacío, 1 = muro, 2 = comida, 3 = vitamina, 5-6-7-8 = fantasmas y 9 = Pacman. Los valores 5-9 indican por tanto que la posición en el mapa es vacía y que en dicha posición está un fantasma o el pacman (ver clase `GameMap` abajo).

Diseño de clases

A continuación se indican las clases y métodos que debes implementar obligatoriamente. A algunos métodos se les da un nombre específico para poder referirnos a ellos en otras partes del texto. Deberás implementar además los métodos (y posiblemente las clases) adicionales que consideres necesario para mejorar la claridad y reusabilidad del código. Cada clase se corresponderá con la definición de un módulo C++ con sus correspondientes ficheros .h y .cpp.

Clase Texture: (Disponible en el CV) Permite manejar texturas SDL. Contiene un puntero a la textura SDL e información sobre su tamaño total y los tamaños de sus frames. Implementa métodos para construir/cargar la textura de un fichero, para dibujarla en la posición proporcionada, bien en su totalidad (método `render`) o bien uno de sus frames (método `renderFrame`), y para destruirla/liberarla.

Clase Vector2D (y Point2D): Representa vectores o puntos en dos dimensiones y por tanto incluye dos atributos (`x` e `y`) de tipo `int`. Implementa, además de la constructora, métodos para consultar las componentes `x` e `y` y para la suma, resta, producto escalar de vectores, producto de un vector por un escalar y operador de igualdad. La suma, resta, productos e igualdad deben implementarse como operadores. En el mismo módulo define un alias para el tipo `Point2D`.

Clase GameMap: Contiene el mapa del juego representado como una matriz dinámica de celdas (de tipo `MapCell`) con sus dimensiones asociadas. Debe proporcionar acceso al juego para leer y escribir en las celdas bien mediante los correspondientes métodos o bien declarando al juego como clase amiga. Define también el tipo enumerado `MapCell` con los valores {`Empty,Wall,Food,Vitamins`}. Observa que el mapa solo contiene información de los elementos estáticos del juego, y no de sus objetos/personajes con movimiento. Esto hace que la carga del mapa desde el fichero la deba realizar el juego, el cual escribirá cada valor en la celda del mapa correspondiente.

Clase Ghost: Contiene al menos las posiciones actual e inicial del fantasma (tipo `Point2D`), su dirección actual de movimiento (`Vector2D`), un puntero a su textura y un puntero al juego. Este último es necesario para que el fantasma conozca su entorno, y en particular, si a la hora de realizar un movimiento, éste es posible (la celda de destino es vacía). Implementa además métodos para construirse, dibujarse (método `render`), actualizarse, es decir, moverse y cambiar de dirección de forma aleatoria (método `update`), y morir (volviendo a su posición inicial).

Clase Pacman: Contiene los mismos atributos que la clase `Ghost` y además la siguiente dirección de movimiento (ver más abajo la funcionalidad de memoria de movimiento), las vidas y la energía restante. Implementa también métodos para construirse, dibujarse (`render`), actualizarse, es decir, moverse y cambiar de dirección cuando sea posible (método `update`), morir (restando una vida y volviendo a su posición inicial), manejar eventos (método `handleEvent`) estableciendo la siguiente dirección cuando el evento es una pulsación de cursor de dirección, y consultar las vidas restantes.

Clase Game: Contiene, al menos, punteros a la ventana y al *renderer*, los tamaños de la ventana y de las celdas, (punteros a) el mapa y el pacman, un array (estático) de (punteros a) fantasmas (usa el tipo `array`), el booleano `exit`, el número de casillas restantes con comida, y el array de texturas (ver más abajo). Define también las constantes que sean necesarias. Implementa métodos públicos para inicializarse y destruirse, el método `run` con el bucle principal del juego, métodos para dibujar el estado actual del juego (método `render`), actualizar (método `update`), manejar eventos (método `handleEvents`), consultar casillas concretas, y determinar la siguiente casilla en una dirección dada (considerando que si sale por un extremo debe aparecer por el otro) devolviendo un booleano indicando si es vacía (método `nextCell`).

Carga de texturas

Las texturas con las imágenes del juego deben cargarse durante la inicialización del juego (en la constructora de `Game`) y guardarse en un array estático (tipo `array`) de `NUM_TEXTURES` elementos de tipo `Texture*`) cuyos índices serán valores de un tipo enumerado (`TextureName`) con los nombres de las distintas texturas. Los nombres de los ficheros de imágenes y los números de frames en horizontal y vertical, deben estar definidos (mediante inicialización homogénea) en un array constante de `NUM_TEXTURES` estructuras en `Game.h`. Esto permite automatizar el proceso de carga de texturas.

Renderizado del juego

En el bucle principal del juego (método `run`) se invoca al método del juego `render`, el cual simplemente delega el renderizado a los diferentes objetos del juego (mapa, pacman y fantasmas) llamando a sus respectivos métodos `render`, y finalmente presenta la escena en la pantalla (llamada a la función `SDL_RenderPresent`). Cada objeto del juego sabe cómo pintarse, en concreto, conoce su posición, tamaño, y textura asociada. Por lo tanto, construirá el rectángulo de destino e invocará al método `render` o `renderFrame` de la textura correspondiente, quién realizará el renderizado real (llamada a la función `SDL_RenderCopy`).

Movimiento del Pacman y de los fantasmas

Los movimientos de los personajes del juego los desencadena el juego y se van delegando de la siguiente forma: el método `update` del juego (invocado desde el bucle principal del método `run`), va llamando a los métodos `update` de cada uno de los elementos del juego, en este caso, pacman y fantasmas. Son ellos los que conocen dónde se encuentran y cómo deben moverse. Para saber si un movimiento concreto en una dirección se puede realizar o no, deben conocer su *entorno*, es decir, deben preguntar al juego (mediante la invocación a su método `nextCell`) si la casilla destino del movimiento está libre. De ahí que los objetos del juego tengan un puntero al juego.

El movimiento del pacman tiene una particularidad que hace el juego más jugable: tiene memoria. Por ejemplo, si el pacman se desplaza a la derecha por un pasillo horizontal y le indicamos que cambie de dirección hacia arriba, no se produce ningún cambio inmediato, pero se moverá hacia arriba cuando llegue a la primera bifurcación que se lo permita. Eso permite al jugador anticipar el siguiente cambio de dirección sin tener que pulsar la tecla en el instante exacto de la bifurcación. Además, si se indica otro cambio de dirección antes de que se haya hecho efectivo el anterior, nos olvidamos del cambio anterior y guardamos el nuevo. Por ejemplo, si el pacman se mueve a la derecha por un pasillo horizontal y se pulsa el cursor ascendente, y antes de cambiar la dirección se pulsa el cursor descendente, el efecto será que irá hacia abajo en el siguiente pasillo descendente. La implementación de esta funcionalidad se llevaría a cabo de la siguiente forma: Cuando se pulsa un cursor, el bucle de manejo de eventos (en el método `handleEvents`) delega el manejo del evento en el objeto pacman (mediante invocación a su método `handleEvent`) el cual guardará la siguiente dirección de movimiento. Cuando el pacman (en su método `update`) pueda tomar la nueva dirección, se hará efectivo el cambio de dirección y se aplicará el movimiento correspondiente.

En esta primera versión los fantasmas se moverán de manera aleatoria, sin importar la posición del pacman. La implementación se lleva a cabo de la siguiente forma: En cada actualización de un fantasma (método `update`) se elige aleatoriamente una de las direcciones posibles. Se excluye la dirección opuesta a la que llevan, para que no se den la vuelta, excepto cuando no queda otra opción (están en un callejón sin salida). Para guardar las direcciones posibles se puede utilizar un vector (clase `vector` de la STL) de pares o direcciones.

Números aleatorios: La generación de números aleatorios en C++ se lleva a cabo mediante la función `rand()` (paquete `cstdlib`) que devuelve un entero aleatorio no negativo. Si se quieren obtener diferentes números aleatorios en diferentes ejecuciones del programa, se debe establecer la *semilla* de números aleatorios con un valor distinto cada vez, por ejemplo mediante la llamada `srand(time(nullptr))` (que solo debe hacerse una vez antes de generar el primer número aleatorio, típicamente durante la fase de inicialización del juego).

Manejo básico de errores

Debes usar excepciones de tipo `string` (con un mensaje informativo del error correspondiente) para los errores básicos que pueda haber. En concreto, es obligatorio contemplar los siguientes tipos de errores: fichero de imagen no encontrado o no válido, fichero de mapa del juego no encontrado, y error de SDL. Puesto que en principio son todos ellos errores irrecuperables, la excepción correspondiente llegará hasta el `main`, donde deberá capturarse, informando al usuario con el mensaje de la excepción antes de cerrar la aplicación.

Pautas generales obligatorias

A continuación se indican algunas pautas generales que vuestro código debe seguir:

- Asegúrate de que el programa no deje basura. Para que Visual te informe debes escribir al principio de la función `main` esta instrucción

  `_CrtSetDbgFlag( _CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF );`

  Para obtener información más detallada y legible debes incluir en todos los módulos el fichero `checkML.h` (disponible en la plantilla en el proyecto `HolaSDL`).
- Todos los atributos deben ser privados excepto quizás algunas constantes del juego en caso de que se definan como atributos estáticos.
- Define las constantes que sean necesarias. En general, no deben aparecer literales que pudiesen corresponder con configuraciones del programa en el código.
- No debe haber métodos que superen las 30-40 líneas de código.
- Escribe comentarios en el código, al menos uno por cada método que explique de forma clara qué hace el método. Sé cuidadoso también con los nombres que eliges para variables, parámetros, atributos y métodos. Es importante que denoten realmente lo que son o hacen. Preferiblemente usa nombres en inglés.

Funcionalidades opcionales

- Extender la mecánica del juego para que el objetivo del juego sea llegar a una determinada puntuación. Para ello se debe establecer qué acciones suman puntuación y cuánta (comer comida, comer vitaminas, comer un fantasma, ...). Cuando se acaba con toda la comida del mapa se pasaría a otra pantalla diferente. Debes mostrar de alguna manera la puntuación actual del juego en todo momento.
- Implementar animaciones en pacman (movimiento de la boca y orientación del movimiento) y en fantasmas (movimiento de las patas y dirección de la mirada). En los fantasmas también se puede visualizar su estado (*hambriento* o *asustado*) como en el juego original. Todo debe quedar bien encapsulado en las clases `Pacman` y `Ghost`.
- Implementa la clase `InfoBar` cuyo método `render` se encargaría de mostrar en la ventana SDL una barra de información del juego que incluya al menos el número de vidas restantes y posiblemente también la puntuación actual.
- Implementar el soporte para permitir guardar y cargar partidas.

Entrega

En la tarea del campus virtual *Entrega de la práctica 1* y dentro de la fecha límite (ver junto al título), uno de los miembros del grupo, debe subir un fichero comprimido (.zip) que contenga la carpeta de la solución y el proyecto limpio de archivos temporales (asegúrate de borrar la carpeta oculta .vs y ejecuta en Visual Studio la opción "limpiar solución" antes de generar el .zip). La carpeta debe incluir un archivo `info.txt` con los nombres de los componentes del grupo y unas líneas explicando las funcionalidades opcionales incluidas y/o las cosas que no estén funcionando correctamente. Ese mismo texto debes subirlo también en el cuadro de texto (sección "texto en línea") asociado a la entrega.

Además, para que la práctica se considere entregada, deberá pasarse una *entrevista* en la que el profesor comprobará, con los dos autores de la práctica, su funcionamiento en ejecución, y si es correcto realizará preguntas (posiblemente individuales) sobre la implementación. Se darán detalles más adelante sobre las fechas, forma y organización de las entrevistas.

Entrega intermedia el 10 de noviembre: Un 20 % de la nota se obtendrá el día 10 de noviembre en función de vuestros avances. Para ello debéis mostrar el funcionamiento de vuestra práctica ese mismo día, bien en la sesión de laboratorio o bien enviando un vídeo de máximo 2 minutos de duración (en el que debéis mostrar los aspectos más relevantes de vuestro código, compilar y mostrar el funcionamiento básico del programa). Para obtener la máxima nota en este entrega se espera que vuestra práctica cargue el mapa y lo visualice junto con el pacman, el cual debe poder moverse por los pasillos usando los cursores de dirección.